03｜强一致性：别再用BASE做借口，来看看什么是真正的事务一致性

2020-08-14 王磊

分布式数据库30讲 进入课程 >

讲述：王磊
时长 19:44 大小 18.08M

你好，我是王磊，你也可以叫我 Ivan。

在上一讲的开头，我提了一个问题：对分布式数据库来说，“强一致性”意味着什么？我们经过分析后得出的结论是这个强一致性，包括数据一致性和事务一致性两个方面。然后，我们介绍了数据一致性是怎么回事儿。那么，今天我们会继续这个话题，谈谈事务一致性。

每次，我和熟悉 NoSQL 同学聊到事务这个话题时，都会提到 ACID 和 BASE。甚至，同学会觉得 ACID 有些落伍了，以 BASE 为理论基础的 NoSQL，才是当下的潮流。

那我们来看看 BASE 是什么？其实，它代表了三个特性，BA 表示基本可用性（Basically Available），S 表示软状态（Soft State），E 表示最终一致性（Eventual Consistency）：

> 基本可用性，是指某些部分出现故障，那么系统的其余部分依然可用。
>
> 软状态或柔性事务，是指数据处理过程中，存在数据状态暂时不一致的情况，但最终会实现事务的一致性。
>
> 最终一致性，是指单数据项的多副本，经过一段时间，最终达成一致。这个，我们在第 2 讲已经详细说过了。

总体来说，BASE 是一个很宽泛的定义，所做的承诺非常有限。我认为，BASE 的意义只在于放弃了 ACID 的一些特性，从而更简单地实现了高性能和可用性，达到一个新的平衡。但是，架构设计上的平衡往往都是阶段性的，随着新技术的突破，原来的平衡点也自然会改变。你看，不用说分布式数据库，就连不少 NoSQL 也开始增加对事务的支持了。

所以说，风水轮流转，今天 ACID 已经是新的后浪了。

事务的 ACID 特性

在数据库中，“事务”是由多个操作构成的序列。1970 年詹姆斯 · 格雷（Jim Gray）提出了事务的 ACID 四大特性，将广义上的事务一致性具化到了原子性、一致性、隔离性和持久性这 4 个方面。我们先来看一下他在 *Transaction Processing Concepts and Techniques* 中给出的定义：

> **Atomicity**: *Either all the changes from the transaction occur (writes, and messages sent), or none occur.*

> **Consistency**: *The transaction preserves the integrity of stored information.*

> **Isolation**: *Concurrently executing transactions see the stored information as if they were running serially (one after another).*

> **Durability**: *Once a transaction commits, the changes it made (writes and messages sent) survive any system failures.*

翻译过来的意思就是：

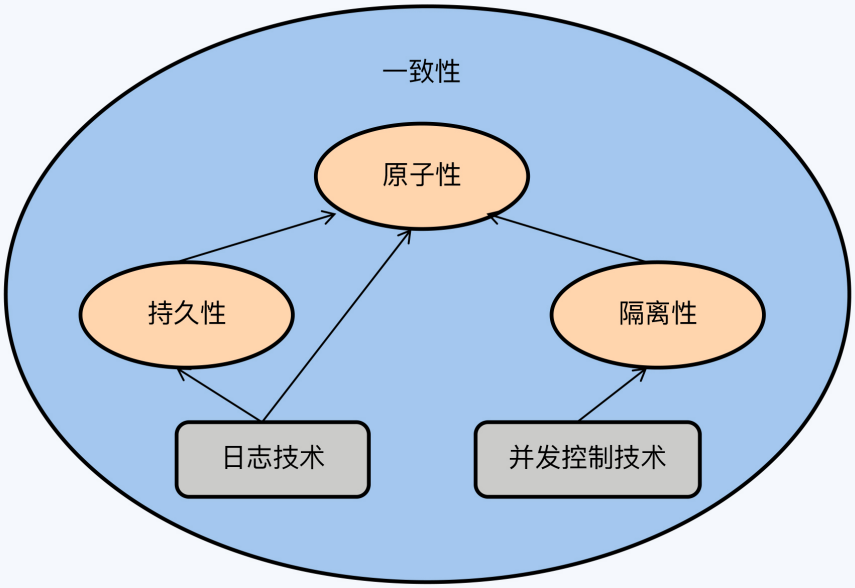
> 原子性：事务中的所有变更要么全部发生，要么一个也不发生。

> 一致性：事务要保持数据的完整性。

> 隔离性：多事务并行执行所得到的结果，与串行执行（一个接一个）完全相同。

> 持久性：一旦事务提交，它对数据的改变将被永久保留，不应受到任何系统故障的影响。

虽然 ACID 名义上并列为事务的四大特性，但它们对于数据库的重要程度并不相同。我用一张图来表示它们的关系。



我们依次来看下。

第一个是一致性，它无疑是其中存在感最低的特性，可以看作是对 “事务”整体目标的阐述。它并没有提出任何具体的功能需求，所以在数据库中也很难找到针对性的设计。

第二个是持久性，它不仅是对数据库的基本要求。如果你仔细琢磨下持久性的定义，就会发现它的核心思想就是要应对系统故障。怎么理解系统故障呢？我们可以把故障分为两种。

1. 存储硬件无损、可恢复的故障。这种情况下，主要依托于预写日志（Write Ahead Log, WAL）保证第一时间存储数据。WAL 采用顺序写入的方式，可以保证数据库的低延时响应。WAL 是单体数据库的成熟技术，NoSQL 和分布式数据库都借鉴了过去。
2. 存储硬件损坏、不可恢复的故障。这种情况下，需要用到日志复制技术，将本地日志及时同步到其他节点。实现方式大体有三种：第一种是单体数据库自带的同步或半同步的方式，其中半同步方式具有一定的容错能力，实践中被更多采用；第二种是将日志存储到共享存储系统上，后者会通过冗余存储保证日志的安全性，亚马逊的 Aurora 采用了这种方式，也被称为 Share Storage；第三种是基于 Paxos/Raft 的共识算法同步日志数据，在分布式数据库中被广泛使用。无论采用哪种方式，目的都是保证在本地节点之外，至少有一份完整的日志可用于数据恢复。

第三个是原子性，是数据库区别于其他存储系统的重要标志。在单体数据库时代，原子性问题已经得到妥善解决，但随着向分布式架构的转型，在引入不可靠的网络因素后，原子性又成为一个新的挑战。

要在分布式架构下支持原子性并不容易，所以不少 NoSQL 产品都选择绕过这个问题，聚焦到那些对原子性不敏感的细分场景。例如，大名鼎鼎的 Google BigTable 甚至是不支持跨行事务的。但是，这种妥协也造成了 NoSQL 的通用性不好。

我们在开篇词就说过，这门课程讨论的分布式数据库是在分布式架构上实现的关系型数据库，那么就必须支持事务，首先就要支持原子性。原子性，在实现机制上较为复杂，目标却很简单，和分成多个级别的隔离性不同，原子性就只有支持和不支持的区别。有关原子性的实现机制，我将在第 9 讲中专门介绍。

最后一个是隔离性，它是事务中最复杂的特性。隔离性分为多个隔离级别，较低的隔离级别就是在正确性上做妥协，将一些异常现象交给应用系统的开发人员去解决，从而获得更好的性能。

可以说，事务模型的发展过程就是在隔离性和性能之间不断地寻找更优的平衡点。我觉得，甚至可以说事务的核心就是隔离性。而不同产品在事务一致性上的差别，也完全体现在隔离性的实现等级上，所以我们必须搞清楚隔离等级具体是指什么。

ANSI SQL-92：对隔离级别最早、最正式的定义

最早、最正式的对隔离级别的定义，是 ANSI SQL-92（简称 SQL-92），它定义的隔离级别和异常现象如下所示：

隔离级别	脏读 Dirty Read	不可重复读 Non-Repeatable	幻读 Phantom
未提交读 Read Uncommited	可能	可能	可能
已提交读 Read Committed	不可能	可能	可能
可重复读 Repeatable Read	不可能	不可能	可能
可串行化 Serializable	不可能	不可能	不可能

SQL-92 定义了四个隔离级别和三种异常现象，这些内容网上很多文章都说得比较清楚，我就不再啰嗦了。如果还不放心，我推荐你去看林晓斌老师的课程《MySQL 实战 45 讲》。

不过，虽然 SQL-92 得到了广泛应用，不少数据库也都遵照这个标准来命名自己的隔离级别，但它对异常现象的分析还是过于简单了。所以在不久之后的 1995 年，Jim Gray 等人发表了论文“A Critique of ANSI SQL Isolation Levels”（以下简称 Critique），对于事务隔离性进行了更加深入的分析。我要特别提示一下，Critique 是数据库领域的经典论文，强烈推荐你阅读原文。

Critique：更严谨的隔离级别

幻读和写倾斜

Critique 丰富和细化了 SQL-92 的内容，定义了六种隔离级别和八种异常现象。其中，我们最关注的是快照隔离（Snapshot Isolation, SI）级别。为什么呢？这是因为在 SQL-92 中可重复读（Repeatable Read, RR）与可串行化（Serializable）两个隔离级别的主要差别是对幻读（Phantom）的处理。这似乎是说，解决幻读问题的就是可串行化。但随着 Critique 的发表，快照隔离被明确提出，这个说法就不适用了，因为快照隔离能解决幻读的问题，但却无法处理写倾斜（Write Skew）问题，也不符合可串行化要求。因为翻译的原因，有时写倾斜也被称为写偏序，都是一个意思。

因此，今天，使用最广泛的隔离级别有四个，就是已提交读、可重复读、快照隔离、可串行化。

而幻读和写倾斜无疑则是通往最高隔离级别的两座大山，那么让我来给你详细解释一下它们到底是什么异常现象。

Critique 对幻读的描述大致是这样的，事务 T1 使用特定的查询条件获得一个结果集，事务 T2 插入新的数据，并且这些数据符合 T1 刚刚执行的查询条件。T2 提交成功后，T1 再次执行同样的查询，此时得到的结果集会增大。这种异常现象就是幻读。

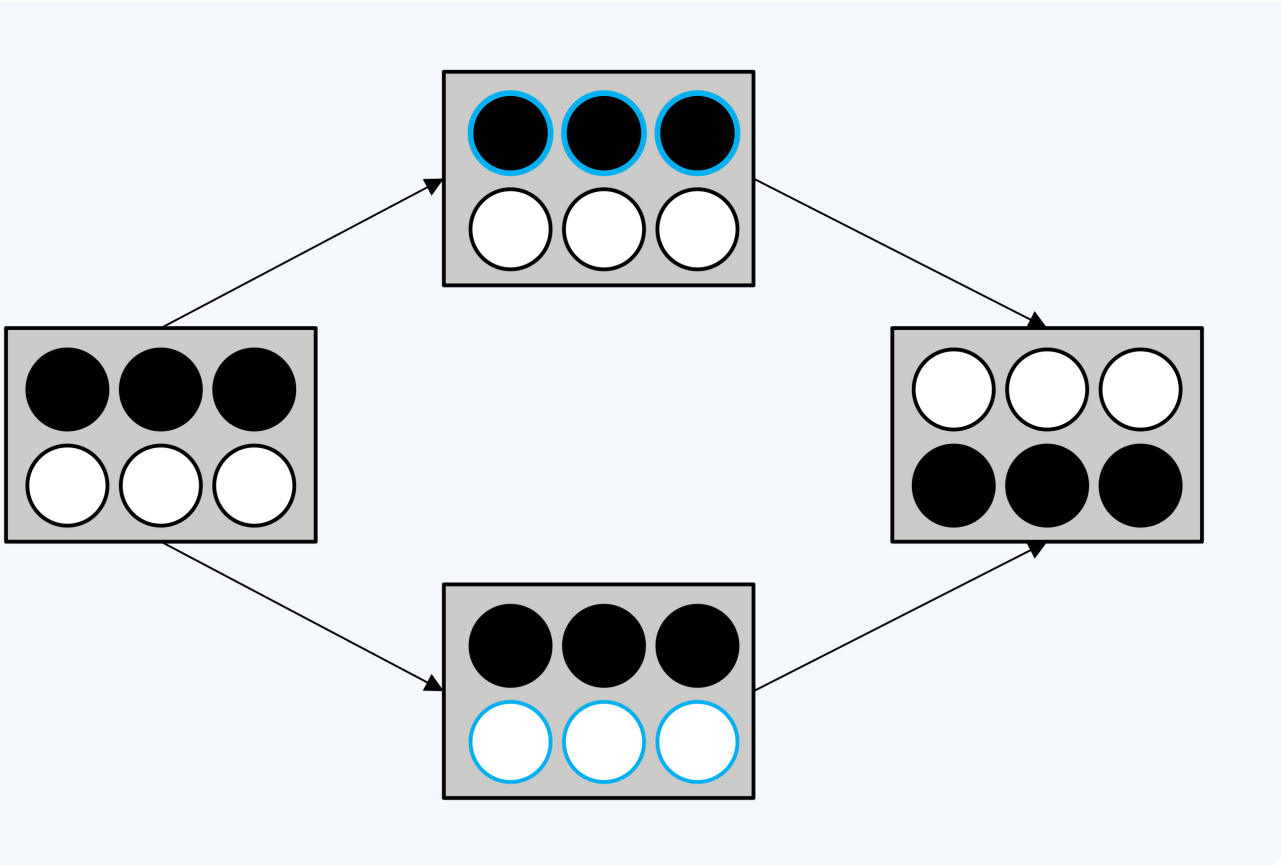
不少人会将幻读与不可重复读混淆，这是因为它们在自然语义上非常接近，都是在一个事务内用相同的条件查询两次，但两次的结果不一样。差异在于，对不可重复读来说，第二次的结果集相对第一次，有些记录被修改（Update）或删除（Delete）了；而幻读是第二次结果集里出现了第一次结果集没有的记录 (Insert)。一个更加形象的说法，幻读是在第一次结果集的记录“间隙”中增加了新的记录。所以，MySQL 将防止出现幻读的锁命名为间隙锁（Gap Lock）。

跟幻读相比，写倾斜要稍微复杂一点，我用一个黑白球的例子来说明。

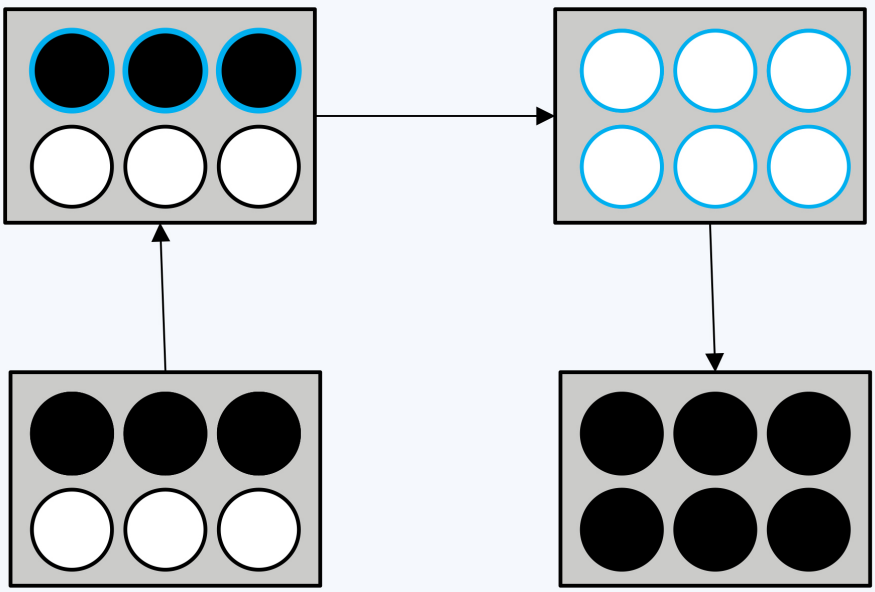
首先，箱子里有三个白球和三个黑球，两个事务（T1,T2）并发修改，不知道对方的存在。T1 要让 6 个球都变成白色；T2 则希望 6 个球都变成黑色。

T1	T2

begin;
update ball set color = white
where color = black;
commit;
	begin;
	update ball set color = black
	where color = white;
	commit;



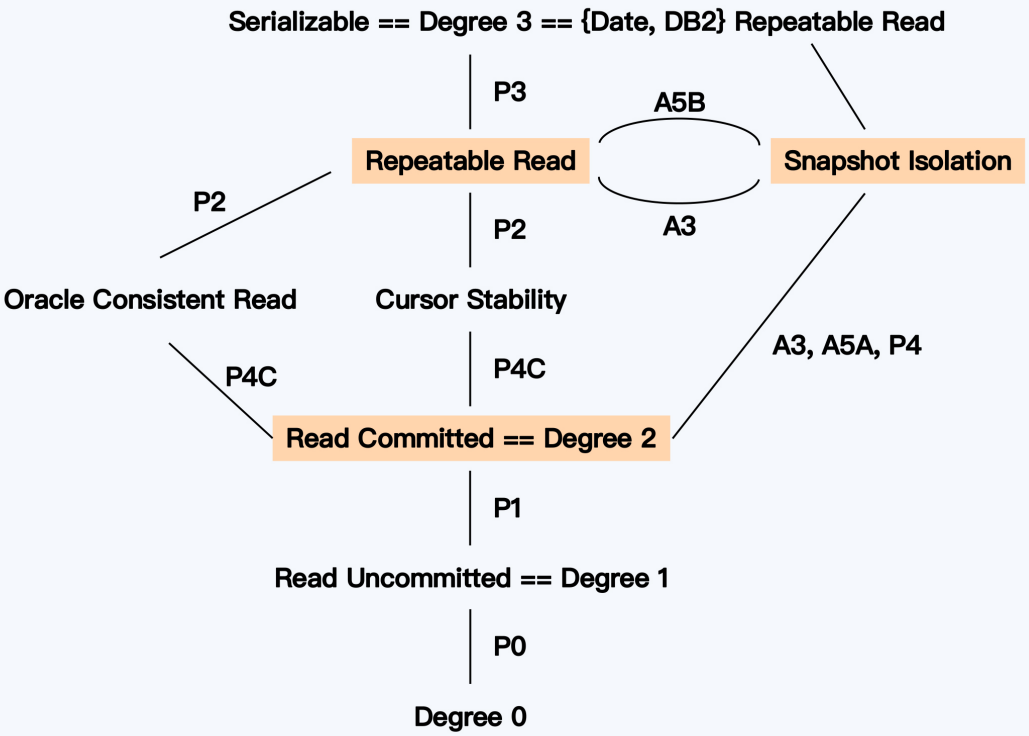
你看，最终的执行结果是，盒子里仍然有三个黑球和三个白球。如果你还没有发现问题，可以看看下面我画的串行执行的效果图，比较一下有什么不同。



如果先执行 T1 再执行 T2，6 个球都会变成黑色；调换 T1 与 T2 的顺序，则 6 个球都是白色。

根据可串行化的定义，“多事务并行执行所得到的结果，与串行执行（一个接一个）完全相同”。比照两张图，很容易发现事务并行执行没有达到串行的同等效果，所以这是一种异常现象。也可以说，写倾斜是一种更不易察觉的更新丢失。

好了，为了让你搞清 Critique 中六种隔离级别的强弱关系以及相互间的差距，我截取了原论文的一张配图。



你可以看到“快照隔离”与“可重复读”在强度上并列，“已提交读”则弱于这两者。事实上，今天大多数数据库支持的隔离级别就在这三者之中。

快照隔离 & MVCC

你可能会问，既然“快照隔离”这么重要，为什么会被 SQL-92 漏掉呢？

这是由于 SQL-92 主要考虑了基于锁（Lock-base）的并发控制，而快照隔离的实现基础则是多版本并发控制（MVCC），很可能是由于当时 MVCC 的应用还不普遍。当然，后来，MVCC 成为一项非常重要的技术，一些经典教材会将 MVCC 作为一种独立的选择，

与乐观并发控制和悲观并发控制并列。其实，在现代数据库中 MVCC 已经成为一种底层技术，用于更高效地实现乐观或悲观并发控制。有了 MVCC 这个基础，快照隔离就成为一个普遍存在的隔离级别了。有关 MVCC 的话题，我会在第 11 讲中继续展开。

隔离性的产品实现

还有一个问题也许你一直想问，为什么不支持最高级别的可串行化呢？

答案可能会让你有点沮丧，那就是在很长一段时间内，学术界都没有找到足够高效的并发控制技术。可能你熟悉的很多数据库声称提供了“可串行化”级别，但这往往只是一种形象工程，因为它们都采用的是两阶段封锁协议，导致性能无法满足生产环境的要求。不过，有些消息让人振奋，虽然不是普适的方案，但少数产品的尝试已经取得进展。

这种尝试来自两个方向。

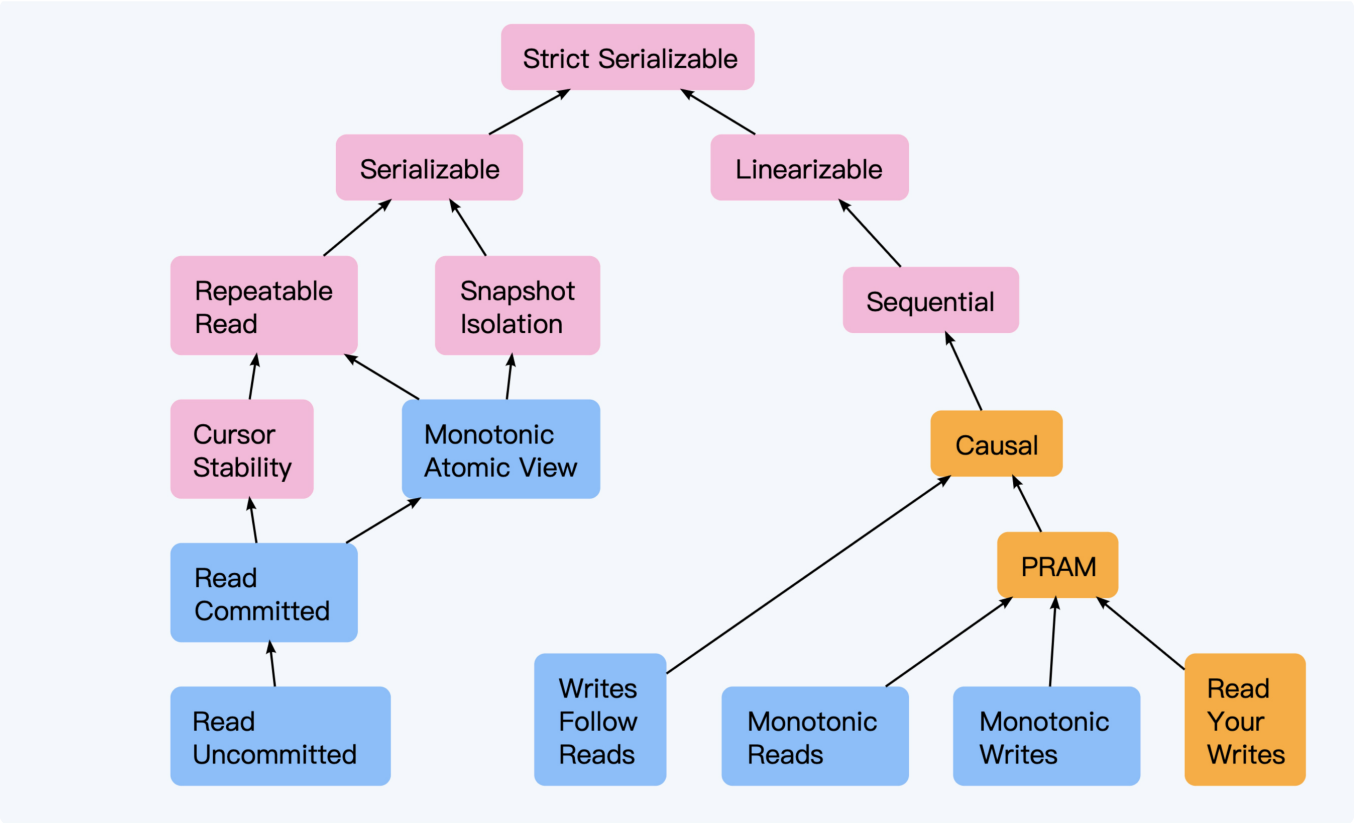
第一个方向是，用真正的串行化实现“可串行化”隔离。我们往往认为多线程并发在性能上更优，但 Redis 和 VoltDB 确实通过串行化执行事务的方式获得了不错的性能。考虑到 VoltDB 作为一款分布式数据库的复杂度，其成功就更为难得了。我想，其中部分原因可能在于内存的大量使用，加速了数据计算的过程。另外，VoltDB 以存储过程为逻辑载体的方式，也使得事务有了更多的优化机会。

如果说第一个方向有点剑走偏锋，那第二个方向就是硬桥硬马了。没错，还是在并发技术上继续做文章。PostgreSQL 在 2008 年提出了 Serializable Snapshot Isolation (SSI)，这实际就是可串行化。而后，兼容 PostgreSQL 生态的 CockroachDB，也同样选择支持 SSI，而且是唯一支持的隔离级别。

这两个方向的尝试都很有趣，我还会在后续的课程中与你深入探讨。

分布式数据库的强一致性

到这里，我们用两讲的篇幅分别介绍了数据一致性和事务一致性，它们共同构成了分布式数据库的强一致性这个概念。我借用一张图来体现三者的关系。



图片原始出处是论文“Highly Available Transactions: Virtues and Limitations”，此处引用的是Jepsen 网站的简化版。

这幅图展现了一个树状结构，左右两个分支上体现事务一致性和数据一致性的各个级别及强弱关系，根节点则体现了分布式数据库的一致性来自两者的融合。图中使用了不同颜色，简单来说，这是区别不同的一致性级别所需付出的性能代价。

对分布式数据而言，最高级别的一致性是严格串行化（Strict Serializable），Spanner 实现的“外部数据一致性”可以被视为与 “Strict Serializable” 等效。但由于两条路径上各自实现难度及性能上的损耗，少有分布式数据库在顶端汇合。即使强大的 Spanner 也提供了有界旧一致性（Bounded Stale），用于平衡性能和一致性之间的冲突。

下面，我总结了一些分布式数据库产品的“一致性”实现情况供你参考。

隔离级别	线性一致性	因果一致性
可串行化 Serializable (SSI)	Spanner	CockroachDB
快照隔离 Snap Isolation (SI)	TiDB	YugabyteDB
可重复读 Repeatable Read (RR)	GoldenDB	
已提交读 Read Committed (RC)	OceanBase 2.0+	

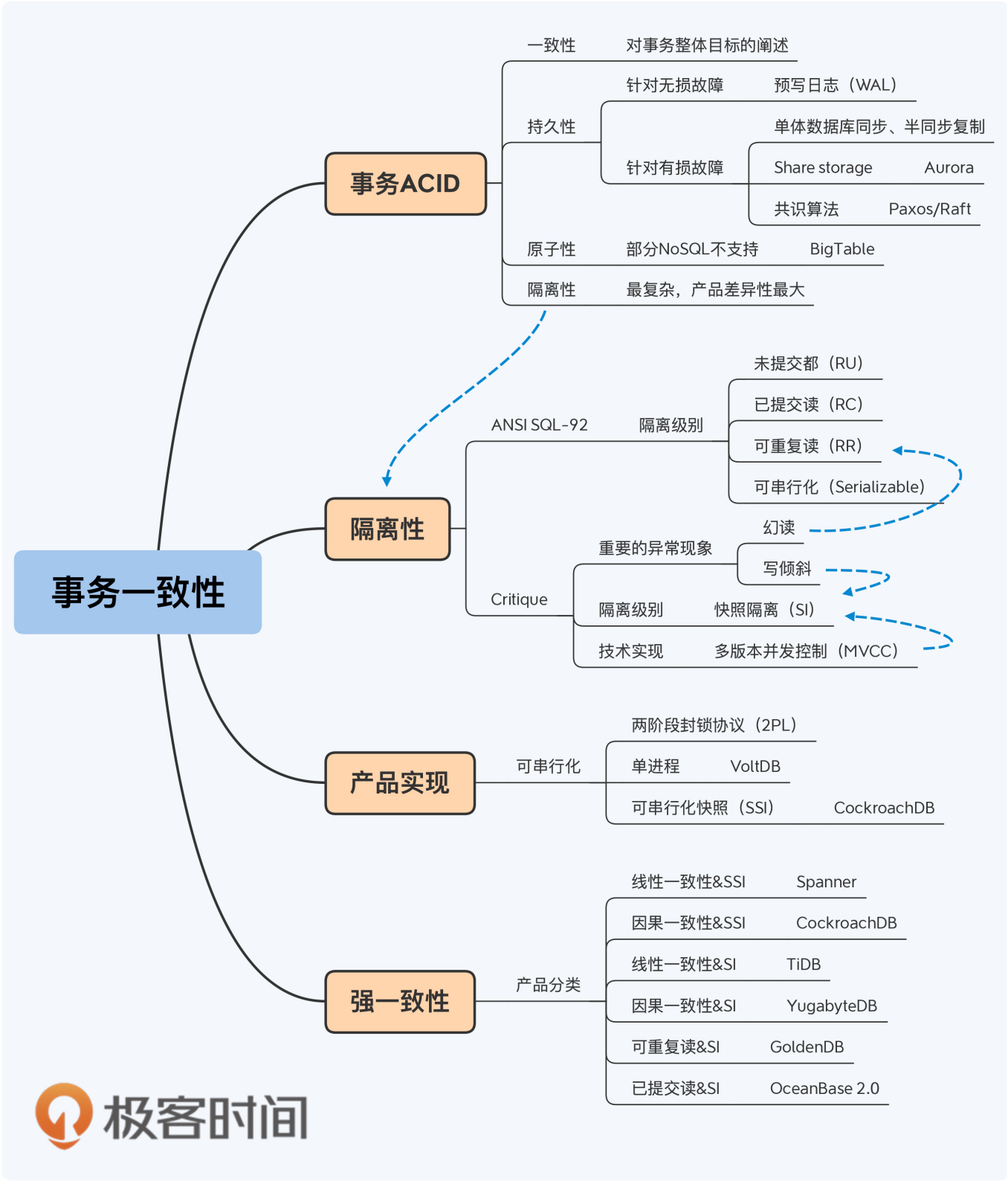
比较特别的是，OceanBase 在 2.2 版本还增加了对“可串行化”的支持，但这是一个被 Oracle 重新定义的“可串行化”，在这个级别 OceanBase 和 Oracle 一样都会出现写倾斜。所以，这不是我们标准的隔离级别，也就没有体现在表格中。

小结

好了，有关事务一致性就讨论到这里，最后让我们来回顾一下今天的重点内容。

1. 数据一致性关注的是单对象、单操作在多副本上的一致性，事务一致性则是关注多对象、多操作在单副本上的一致性，分布式数据库的一致性是数据一致性与事务一致性的融合。
2. 广义上的事务一致性被细化为 ACID 四个方面，其中原子性的实现依赖于隔离性的并发控制技术和持久性的日志技术。
3. 隔离性是事务的核心。降低隔离级别，其实就是在正确性上做妥协，将一些异常现象交给应用系统的开发人员去解决，从而获得更好的性能。所以，除“可串行化”以外的隔离级别，都有无法处理的异常现象。
4. 研究人员将隔离级别分为六级，你需要重点关注其中四个，分别是已提交读、可重复读、快照隔离、可串行化。前三者是单体数据库或分布式数据库中普遍提供的，可串行化仅在少数产品中提供。

好了，到这里，加上前一节“数据一致性”，我们用了两讲阐述了分布式数据“强一致性”的含义。在严格意义上，分布式数据库的“强一致性”意味着严格串行化（Strict Serializable），目前我们熟知的产品中只有 Spanner 达到了这个标准，其同时也带来了性能上的巨大开销。如果我们稍稍放松标准，那么“数据一致性”达到因果一致性且“事务一致性”达到已提交读，即可认为是相对的“强一致性”。还有一点非常重要，分布式数据一致性并不是越高越好，还要与可用性、性能指标结合，否则就成了形象工程。



思考题

课程的最后，我要留给你一道思考题。

我们在事务持久性部分提到了预写日志（WAL），它可以保证在系统发生故障时，数据也不会丢失。但是，如果写日志成功，而写数据表失败，又要如何处理呢？你可以根据自己的经验，讲讲该如何设计这个过程吗？

欢迎你在评论区留言和我一起讨论，我会在答疑篇回复这个问题。如果你身边的朋友也对事务一致性这个话题感兴趣，你也可以把今天这一讲分享给他，我们一起讨论。

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 02 | 强一致性：那么多数据一致性模型，究竟有啥不一样？

下一篇 04 | 架构风格：NewSQL和PGXC到底有啥不一样？

精选留言 (9)

写留言

赵见跃
2020-08-14

写的太好了！！！
当初就是学习了林晓斌老师《MySQL 实战 45 讲》，使我认识了极客时间，课程非常好！随后又买了几个课程，可惜都没有达到林晓斌老师的高度，曾经一度对极客时间产生了怀疑，
还好，幸运的是今天看到了王磊的课程，感觉又有希望了。王老师的课程客观理性、深...
展开

作者回复: 谢谢，这个评价好高呀，写起来更有动力了，哈哈。

8 6

与路同飞
2020-08-14

老师每节课的思维导图很赞

展开

作者回复: 希望能帮大家高效得学习课程的内容：)

2

distdev
2020-08-14

关于幻读， 一直不明白实际工程中幻读在哪种情况下会有问题？似乎绝大多数情况一个事务看见其他的事务的创建或者删除记录都不是问题。老师能否给举个例子？

1

南国
2020-08-14

首先回答问题，其实对此方面了解不多，强答一下：其实预写日志在我看来就是redo日志，如果redo日志已经成功证明已经落盘，此时数据可以根据redo日志异步的刷回磁盘，写数据表失败应该就是后面异步写回出现问题，我们只需要重演redo日志就可以了。当然这只是最简单的过程，细节还是看其他留言和老师的解答啦

...

展开

1

一步
2020-08-16

对于文章那个写倾斜的图不是很理解，老师可以解释下执行过程吗？ 方便更好的理解

AA
2020-08-15

老师，citus可以加入讲讲么

展开

作者回复: 你好，抱歉，我对citus确实没什么研究，这似乎是一种数据库中间件.....

扩散性百万咸面包
2020-08-15

我之前的发言有些误解，我再阐述一下：
SI隔离级别是MVCC，实际上RR也可以用MVCC，不过之前没有这种技术，都用的是2PL。SI主要通过Gap lock来解决RR的幻读？因为光一个MVCC是解决不了幻读的。

展开

扩散性百万咸面包
2020-08-15

有点问题：
快照隔离相当于是比可重复读解决了幻读的问题，文章中说是MVCC的功能特性，但是MVCC并不能解决幻读呀，真正解决幻读的是Gap Lock（Mysql）？

而且可重复读也可以使用MVCC来实现吧？

展开

1

distdev
2020-08-14

关于redo log有一个问题， 就是何时同步到磁盘？完全同步太慢，如果成批处理又有可能丢数据， 老师能否谈谈经验？

展开

作者回复: redo log是接到客户端请求后立即落盘的，它的同步方式包括，异步复制，半同步复制和全复制，一般如果对RPO有要求选择半同步复制会多些。